TM-1563/014/01

SPAN REFERENCE MANUAL

SPAN Data-Transformations and Stratification Capability

4 March 1965

TECHNICAL MEMORANDUM

(TM Series)

SPAN REFERENCE MANUAL

SPAN Data-Transformations and Stratification Capability

by

Vladimir V. Almendinger

4 March 1965

SYSTEM DEVELOPMENT CORPORATION
2500 COLORADO AVE.
SANTA MONICA
CALIFORNIA

*TM-1563/014/00, a preliminary version of this document, was not published.

(Page ii Blank)

ABSTRACT

This document is one of a series of volumes of the SPAN REFERENCE MANUAL describing the SPAN system for data management and statistical analysis. The SPAN system provides an integrated file processing and multivariate analysis capability for problems involving large data matrices with large numbers of variables. The system is oriented to problems of social science in general and urban data analysis in particular.

This volume describes the specification and use of SPAN capabilities for the transformation of the contents of a data file in the course of its input to a SPAN process. The data-transformation capability allows the user to state algorithmic procedures leading to the generation of new data variables as transformations of the source data. The entity stratification capability provides for the classification and selection of entity records according to user-specified criteria.

(Page iv Blank)

FOREWORD

Capabilities, specifications formats, and other details of the application of the data-transformation and entity stratification languages described in this volume correspond to the following operational components of the SPAN system:

1. SPAN System Library Version 3 (and subsequent versions); and

2. SPAN System Supervisor (binary deck), dated February 12, 1965.

The current operational status of the SPAN system is documented in TM-1563/050.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

I. INTRODUCTION: THE STANDARD INPUT PROCESS

A typical single SPAN task consists of obtaining information from a source file and subjecting it to statistical analysis, tabulation, listing, or other forms of processing. SPAN provides the capability to transform or stratify data taken from the source file prior to subsequent processing. The methods for accomplishing this are described in this volume.

As a simple example, the source file might contain data on the total population and its age distribution by census tracts. The data-transformation capability makes it possible to compute the new variable "percent population over 65" for each census tract. The stratification capability makes it possible to stratify the tracts into groups according to size of population. However, the stratification process takes place after data transformations so that one could stratify on the basis of a newly created variable. Thus, in terms of the above example, one could stratify the tracts into groups according to percent population over 65. Inherent in the stratification process is the capability to select or reject tracts on the basis of specified criteria. The wide range of possibilities in both data transformations and stratification is discussed in the following sections.

The SPAN capabilities for data transformations and entity stratifications (including record selection) may be exercised in the following SPAN modules:

Factor Analysis
Regression Analysis
Latent Class Analysis (record selection only)
Rectangular Product Moments
File Transformation
STARS Files Abstractor
STARS Files Collator
STARS Files Tabulator
STARS Files Summary-Sort
SPAN Report Generator
System for Mixed Data Structures Reduction
SPAN Graphic Display system

THE SOURCE FILE

Only those aspects of file organization necessary to an understanding of this volume will be covered here; a complete discussion of file organization is given in TM-1563/015/xx. Furthermore, the discussion will be limited to "simple files"; a simple file is one in which each record has the same format. "Mixed data files" and their treatment are discussed in TM-1563/023/xx.

The typical source file consists of observations on the properties of entities. Entities might be census tracts, individuals, time intervals, events, or any other "units"; properties might be population, land area, age, percent unemployed, and so forth. One can think of the file as a two-dimensional array; in SPAN, by convention, rows are entities, and columns are properties. A typical source file is made up of records, each record consisting of the values of the properties for one entity.

The properties for a given entity are of two types: codes and variables. Codes are properties that identify the entity, such as census tract code, man number, X-Y coordinates, event identification, etc. A typical source-file record permits a total of fifty different codes. Variables are those properties such as population, age, etc., that represent the computational data for the entities. Typically, a maximum of 1000 variables is possible. Within a source file record, the set of codes is logically separate from, and precedes, the set of variables.

THE STANDARD INPUT PROCESS

A file is processed serially, that is, a single record is obtained, transformed, stratified, and subjected to such further processing as specified, and then another record is obtained. For ease of discussion, we shall define two internal regions, the I-region and the X-region. As each record is obtained from the source file, the codes are placed in the I-region and the variables in the X-region.

Obviously the user specifies whether a property is a code or a variable. However, codes and variables, as will be seen, differ in the manner in which they can be operated upon, and sometimes the user may find it necessary to include the same property both as a code and as a variable.

Each code or variable corresponds to one location in the I-region or X-region respectively. (The internal representation of codes and variables will be discussed later.) The I-region can accommodate 100 codes; since the typical record contains, at most, 50 codes, additional space is available for new codes arising from data transformations. Codes are identified by their location in the I-region, e.g., I(1), I(2), ..., I(100). Codes I(99) and I(100) are sometimes used for specific purposes by SPAN modules, and are therefore not always available. The X-region normally accommodates 1400 variables, identified by their location, e.g., X(1), X(2), etc. The typical record contains a maximum of 1000 variables; additional space is thus available for new variables arising from transformations. Note that if the record contained only 200 variables, locations X(201) through X(1400) would be available; a similar comment applies to the I-region. The I-region and X-region are automatically cleared prior to the processing of the first record.

Referring to Figure I-1, we see that a record is taken from the source file, and the codes and variables are placed, respectively, in the I-region and X-region. If any data transformations or stratification are to take place, or any subsequent computations are to be made, as distinct from merely moving data, all of the variables are automatically converted to floating point, if they are not already in that form.

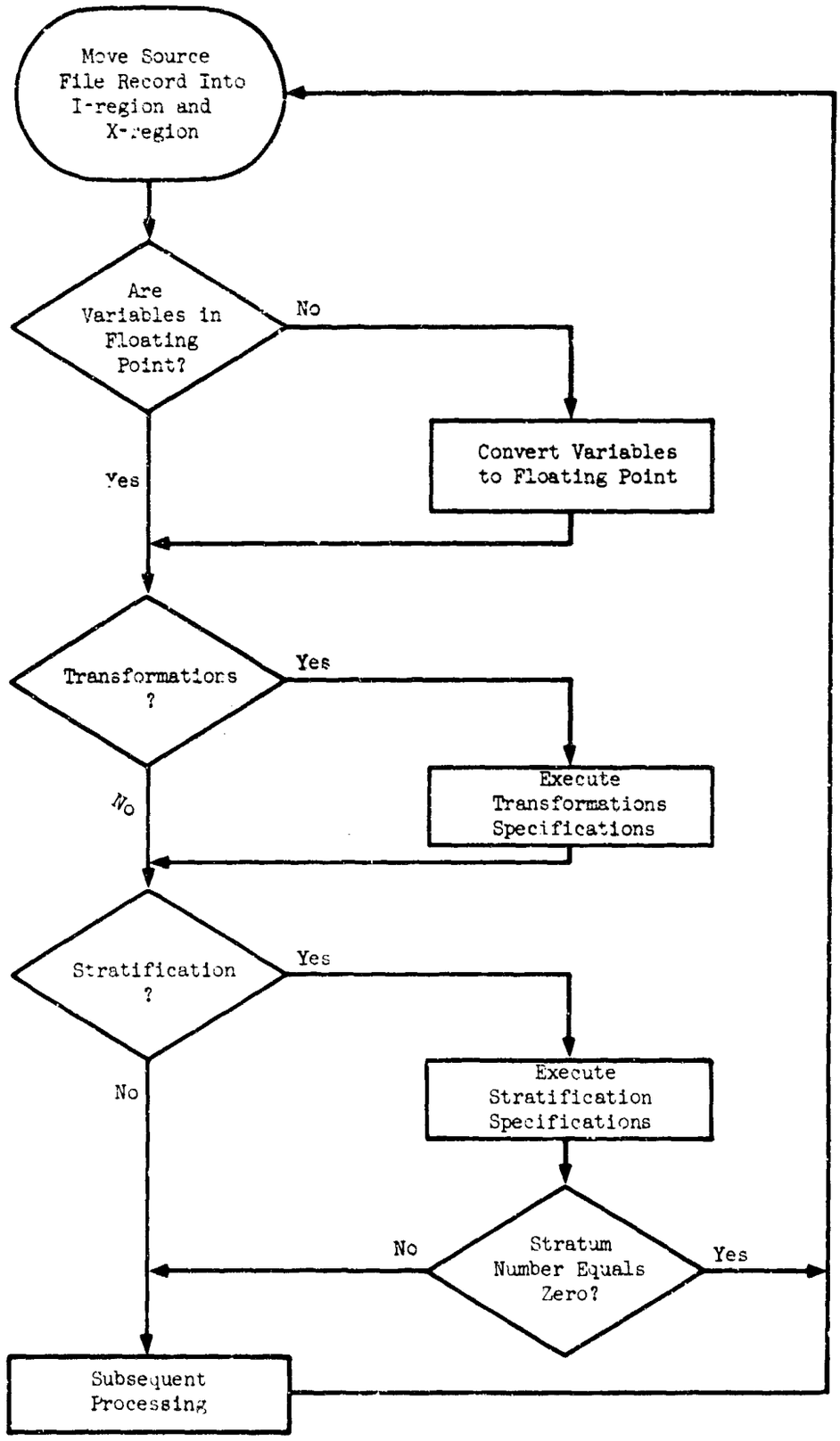
Data transformations may be made on variables or codes. Transformations on variables are normally floating-point arithmetic operations; that is, new variables are created by arithmetic operations on existing variables. Codes may be transformed by means of Boolean or integer operations.

Stratification permits the user to assign stratum numbers to each entity according to a wide range of user-specified conditions. Thus, if one wishes to stratify a file of records into five groups according to the value of a given variable, one might assign the stratum numbers 1, 2, ... 5. As the stratification specifications are executed, the stratum number for each record is automatically placed in I(99) and is available, if desired, in subsequent processing.

If a record does not meet the specified conditions, the stratum number is automatically set at zero, and such a record is rejected; i.e., it does not undergo subsequent processing. Thus, selection is inherent in stratification, and as a special case, one can select any particular records, and thus reject others, by assigning any non-zero integer as the stratum number to the records to be selected.

Each record from the source file undergoes the processes shown in Figure I-1, and the resulting records can be thought of as making up a derivative file. It is this derivative file that is the "input" for subsequent processes. Most SPAN-specific processes, such as abstracting, summarizing, and collating, or calculating a correlation matrix, are specified with respect to the derivative file. This transformation of source file into derivative file is known as the standard input process. If no transformations or stratification are specified, the derivative file will be identical to the source file. Implications of the standard input process for identifiers of file content, such as labels of variables or names of the codes, variables, and entities sets in a STARS file, are discussed under the heading of STARS output file description in the SPAN REFERENCE MANUAL volume on "Data File Manipulation and Processing," TM-1563/021/xx.

In the following two chapters the rules for specifying data transformations and entity stratification are discussed. The reader should assume that the data are already in the I and X-region and that he knows the positions of the various codes and variables.



THE STANDARD INPUT PROCESS

Figure I-1

II. DATA TRANSFORMATIONS

Data-transformation specifications are used to create new codes or variables.* Newly created codes and variables can be used in subsequent processing and to create additional codes and variables. Data-transformation specifications are executed on each entity record, as described in Chapter I. The rules for specifying data transformations are discussed in this chapter.

Data-transformation specifications are composed of an arbitrary collection of statements, of which there are three types: (1) transformation statements, used to specify arithmetic computations; (2) condition statements, used to control the sequence in which statements are executed; and (3) special statements. Statement are made up of expressions and these, in turn, of terms. These and other elements of the data-transformation specification are discussed below.

TERMS

A term consists of an operand prefixed by its operator. There are three types of terms: (1) arithmetic terms, (2) Boolean terms, and (3) integer terms. The type of term is uniquely determined by the combination of operator and operand; the same symbol is often used for operators in different types of terms, even though it represents a different operation.

Figure II-1 lists the various possible operands. It will be recalled that X(15) refers to the 15th location in the X-region, which corresponds to the 15th variables; I(23) refers to the 23rd location in the I-region, which corresponds to the 23rd code. Note that in transformations text constants are always enclosed in parentheses.

---

* The SPAN data-transformation language described in this chapter was conceived and implemented by Robert A. Hoodes, who participated in various phases of SPAN development. The principal virtue of the notation is its easy translatability, leading to a simple but highly efficient execution method. An early version of the language was first used by Mr. Hoodes in his IBM 704 MUSP system for multivariate analysis. Mr. Hoodes is now with the IBM Federal Systems Division.

TYPES OF OPERANDS		
Symbol	Name	Example
V	Floating point variable	X(15)
F	Function of variable	SQRT X(15)
D	Decimal number constant	(-.35)
C1	Decimal integer code	I(23)
C2	Octal integer code	I(23)
C3	Alphameric code	I(23)
C4	Complement of octal integer code	-I(15)
K1	Decimal integer constant	(11)
K2	Octal integer constant	(Ø77077)
K3	Alphameric constant	(HA5PX43)

Figure II-1

VARIABLES AND FUNCTIONS OF VARIABLES (V or F)

General Form
A variable is represented by X(n), where n is the index of its location in the X-region.

Variables, such as X(65), are represented internally as floating-point numbers; they are automatically converted to decimal representation for printout. The magnitude of a floating-point variable must lie approximately between 10^{-38} and 10^{38}. Functions of variables, such as SIN X(28), are floating point numbers. Figure II-2 lists all defined functions.

FUNCTIONS OF VARIABLES	
Square root	SQRT X(90)
Logarithm (natural)	LØQ X(90)
Exponential	EXP X(90)
Sine	SIN X(90)
Cosine	CØS X(90)
Arctangent	ATAN X(90)

Figure II-2

The function of a variable is a single operand. Where the function of a variable would not yield a real number of allowable magnitude, as in $(-4)^{\frac{1}{2}}$, log 0, or e^{90}, the value of the function is set to -0.

DECIMAL NUMBER CONSTANTS (D)

General Form
A decimal number constant consists of a string of decimal digits with a decimal point at the beginning, at the end, or between two digits.

The decimal number constant may be signed or unsigned. A decimal number is represented internally as a floating-point number. The magnitude of a decimal number constant must lie between 10^{-14} and 10^{14}, or be zero. Examples: (-.15), (3.1416), (12.) or (0.).

DECIMAL INTEGERS (C1 or K1)

General Form
A decimal integer constant consists of 1-5 decimal digits written without a decimal point. A decimal integer code is represented by I(n), where n is the index of its location in the I-region.

A decimal integer may be signed or unsigned. Its magnitude must be less than 2^{15}. A decimal integer constant would be expressed as (1492), (-2), or (0). A decimal integer code is represented by its address, for example, I(92). Decimal integers are stored internally as pure binary numbers, occupying the left-half word, with the right 18 binary bits being all zeros. If a decimal integer is negative, the minus sign occupies the high-order position of the binary word; this must be remembered when using negative integers in Boolean operations.

OCTAL INTEGERS (C2 or K2)

General Form
An octal integer constant consists of 1-12 octal digits preceded by the letter O. An octal integer code is represented by I(n), where n is the index of its location in the I-region.

The maximum value of an octal integer is 2^{36}. Leading zeros need not be indicated. Examples: (Ø356), (Ø777700007777), (Ø3). An octal integer code is represented by its address, for example, I(92). Octal integers are stored internally as pure binary numbers, each octal digit comprising three binary bits.

ALPHAMERIC CONSTANTS AND CODES (C3 or K3)

General Form
An alphameric constant consists of 1-6 alphabetic and/or numeric characters preceded by the letter H. An alphameric code is represented by I(n), where n is the index of its location in the I-region.

An alphameric constant may not include special characters. The H to indicate mode must be used even if word begins with an H. Examples: (HABC), (H2376), (H23KV6), (HHAPPY). If the string exceeds six characters, only the six right-most characters are used. For example, (HALPHAMERIC) is equivalent to (HAMERIC), i.e., the last six characters.

An alphameric code is represented by its address, for example: I(92). Alphameric words are stored internally as binary numbers, with two octal digits, and therefore six binary bits, for each character. They are left-justified, so that the right characters consist of blanks. The IBM 7090 octal (storage) representation of alphameric characters is given in Appendix A , "7090 Representation of Alphameric Characters."

COMPLEMENT OF OCTAL INTEGER CODE (C4)

The 1's complement of an octal integer code is indicated by placing a ["-"] symbol ahead of the address of a code, for example, -I(15). The combination represents a single operand.

ARITHMETIC TERMS

Arithmetic terms may be simple or complex. A simple arithmetic term consists of one operator and one operand. A complex arithmetic term represents a single operation performed on more than one operand. Figure II-3 lists the operators that may be used in simple arithmetic terms; Figure II-4 lists the two types of complex terms. Only certain operands may be used with a given operator; these are indicated in Figures II-3 and II-4 by means of the operand symbols defined in Figure II-1.

SIMPLE ARITHMETIC TERM OPERATORS		
Operator	Operation	Applicable to Operand Types
+	Add	V, F, D
-	Subtract	V, F, D
*	Multiply	V, F, D
/	Divide	V, F, D
TØ	Store and proceed	V
=	Store and clear	V

Figure II-3

Division by zero will set the quotient to -0. The difference between the two "store" operators will be discussed in the section on expressions.

COMPLEX ARITHMETIC TERMS		
Term	Operation	Admissible Operand Types
SUM x_1 THRU x_2	Sum operands x_1 through x_2	V
MED x_1 THRU x_2 INT x_3	Calculate median over variables x_1 through x_2, with x_3 being the first element of a set of the lower limits of corresponding category intervals; if n is the number of categories, the (n+1)th element contains the upper limit of the nth interval.	V

Figure II-4

A detailed example of a median computation will be given later.

BOOLEAN TERMS

Boolean terms consist of one operator and one operand. They are listed in Figure II-5. Only certain operands may be used with a given operator; these are indicated in Figure II-5 with the operand symbols defined in Figure II-1.

BOOLEAN TERM OPERATORS		
Symbol	Operation	Applicable to Operand Types
+	logical-or	All C and K
*	logical-and	All C and K
/	exclusive-or	All C and K
LEFT	shift left n bits	K1
RIGHT	shift right n bits	K1
TØ	store and proceed	All C
=	store and clear	All C

Figure II-5

For Boolean operations, including shifts, recall that all operands are internally represented as pure binary quantities of 36 binary bits. Shifts are, in effect, accumulator shifts, not "long" or "logical" shifts; therefore bits are lost in this process. On a left shift of n bits, (n-1) bits are lost, while n bits are lost on a right shift of n bits. Vacated positions are filled with zeros.

INTEGER TERMS

Integer terms consist of an operator and an operand. The operators, and applicable operands, are given in Figure II-6.

INTEGER TERM OPERATORS		
Symbol	Operation	Applicable to Operand Types
+X	Integer Add	C1
-X	Integer Subtract	C1
*X	Integer Multiply	C1
/X	Integer Divide	C1
=X	Store and Clear	C1

Figure II-6

EXPRESSIONS

An expression is a string of terms, none of which contains the operator [=] or [=X]. Each line below is an example of one expression:

Expression	Type
X(52) - X(43) / (2.84)	Arithmetic
I(2) * (Ø770000) LEFT (18) + (HØUKE)	Boolean
(.55) - SQRTX(55) - X(12)	Arithmetic
(Ø707) / I(87) + -I(3)	Boolean
XI(52) /X I(33) *X I(1)	Integer
MED X(14) THRU X(21) INT X(38) TØ X(80) * (2.)	Arithmetic

Blanks are ignored in expressions, so that expressions equivalent to the above can be written in many ways. In the above examples, each of the following represents a single term: -X(43); LEFT (18); - SQRTX(55); - -I(3); /X I(33); TØ X(80); MED X(14) THRU X(21) INT X(38).

RULES FOR CONSTRUCTING EXPRESSIONS

1. All terms in an expression must be of the same type; e.g., all arithmetic terms, Boolean terms, or integer terms. Mixed expressions are not detected by the translator and will yield incorrect results.

2. A complex arithmetic term may appear only as the leading term. Thus the following is a correctly formed expression,

   SUMX(35) THRU X(40) * X(2)/(100.)

   while the following expression is in error,

   X(2) * SUMX(35) THRU X(40)/(100.)

3. For expressions made up of either simple arithmetic or Boolean terms the operator of the leading term is always an implicit [+] . That is, no operator except the [+] may be used for the leading term, and the [+] is never written. (See the previously given examples.)

4. Expressions are evaluated from left to right; the operator symbol in each term defines the relation between the result of preceding terms and the value referred to by the operand. In the expression X(52) - X(43) / (2.84) the quantity X(43) is subtracted from X(52) and the result is then divided by (2.84). If one wanted to compute X(43) divided by (2.84) and the quotient to be then subtracted from X(52) one would need to first perform the division and then store the quotient in a temporary location, as will be discussed in the section on Statements. In some cases, other methods can be used; thus, if the operator ahead of X(43) were a + instead of a -, one could write X(43) / (2.84) + X(52). In general, if $\oplus$ is a typical operator and x is a typical operand, then the expression

   $$x_1 \oplus_2 x_2 \oplus_3 x_3 \oplus_4 x_4$$

   is evaluated as if it read

   $$(((x_1 \oplus_2 x_2) \oplus_3 x_3) \oplus_4 x_4)$$

   However, parentheses may not be used for this purpose in an expression.

TRANSFORMATION STATEMENTS

General Form
A transformation statement is an expression terminated by a term whose operator is [=] or [=X].

Examples of transformation statements are:

```
X(52) - X(43) / (2.84) = X(3)
I(2) * (Ø770000) LEFT (18) + (HØUKE) = I(2)
X I(2) +X I(2) =X I(4)
```

The [=] and [=X] operator stores the result of the expression in the location represented by the operand, that is, in a location in the I-region or X-region. A term whose operator is [=] or [=X] is not part of the expression. A term whose operator is [TØ] or [TØX] is part of an expression. It stores the results of the previous terms into the location represented by its operand. The results are also immediately available for the next term. Thus, in the statement

```
X(1) + X(2) TØ X(3) + X(4) = X(5)
```

the sum of X(1) + X(2) is stored in X(3), is then added to X(4), and the final result stored in X(5). However, if the total stored in X(5) is to be used again, it is obtained from storage by explicitly indicating X(5) in a subsequent statement. All of the terms in a statement must be of the same type.

Effect of Transformation Statements on Variable Labels

A source-file variable may be the operand of a [TØ] or [=] operator. It should be noted, however, that whenever the value of a source variable is replaced in this manner by a result of data transformations, the label associated with the source variable is set to zero, and the source variable is no longer available for further processing. Procedures for associating labels with newly computed variables are discussed in TM-1563/021/xx.

STATEMENT LABELS

General Form
A statement label is a string of, at most, six alphanumeric characters bracketed by $ symbols.

Any statement may be assigned a label. For example:

$ A51 $; $ NEXT $; $ 111111 $.

The label is placed ahead of a statement thus

$ NOW $ X(1) - X(2) = X(3).

Blanks in the label, as elsewhere in the transformations text, are ignored. A label may stand alone at the end of the transformations text to permit a terminal transfer to the end of specifications.

CONDITION STATEMENTS

General Form
Condition statements are of the form $a_1 \bullet a_2$ OR TO s where a_1 and a_2 are expressions, $\bullet$ is a condition operator, and s is a statement label.
Condition statements make it possible to control the sequence in which statements are executed.

If the condition asserted by the condition statement is true, transfer is made to the statement identified by the label. Otherwise, the statement following the GØ TØ command is executed. Integer terms are not permitted in the condition statement. In a condition statement a complex arithmetic term (for definition, see Figure II-4) may appear only as the leading term of the left-hand expression. Both expressions in a condition statement must contain the same type of terms. A list of condition operators is given in Figure II-7.

CONDITION OPERATORS	
Symbol	Condition Asserted
EQ	Equal
GR	Greater Than
LS	Less Than
GQ	Greater or Equal
LQ	Less or Equal

Figure II-7

Examples of condition statements are:

```
X(3) + (3.0) GR X(25) - LØG X(13) GØ TØ $ NEXT $
X(5) EQ (3.) GØ TØ $ 3 $
I(3) EQ (HPHILAD) GØ TØ $ AGAIN $
I(5) * (ØTTTTOOOOTTTT) EQ (HABOOCD) GØ TØ $ SMILE $
```

If the values of two expressions made up of arithmetic terms are compared, zero is greater than minus zero. If the values of two expressions made up of Boolean terms are compared, zero is less than minus zero, since the minus sign is the high-order bit.

SPECIAL STATEMENTS

There are two special statements: the unconditional transfer and the weighting function.

Unconditional Transfer Statement

General Form
GØ TØ s where s is a statement label.

For example, the statement

GØ TØ $ HERE $

causes the statement labeled HERE to be executed next.

Weighting Function

General Form
WGHT x_1 THRU x_2 BY x_3 where x_1 and x_2 are respectively the first and the last of a range of variables to be weighted, and x_3 is a multiplier variable.

The weighting function provides for the multiplication of each of a set of X-regio variables by another variable. For example, the statement

WGHT X(5) THRU X(12) BY X(40)

results in replacement of each variable from X(5) through X(12) by its value multiplied by the value contained in X(40).

Effect of the Weighting Function on Variable Labels

Although the weighting function may be used to modify the values of source variables, labels associated with the source variables are not affected by this process. If the weighting function is used to define new variables, the operands of the weighting function must appear elsewhere in the transformations specifications also as operands of the [TØ] or [=] operators.

HOW TO SPECIFY DATA TRANSFORMATIONS IN A SPAN JOB

Data-transformation specifications are composed of an arbitrary collection of transformation statements, condition statements, and special statements that are stated in the particular order in which they are to be evaluated. Data-transformation specifications that are to operate on the data input to a particular SPAN job must be included as a separate sentence among the control specifications for that job. In a control sentence, the specifications text takes the form of a string predicate of an appropriate control word; that is, the text is delimited by apostrophies and preceded by a transformations-declaring control word.

General Form
TRANSForm F-TRANsform Ø)TRANsform } s where s is the data-transformation specifications text (S6000), and Ø stands for source-file number in the case of multifile input (Ø = 1,2,3 in the STARS Files Collator).

A simple example of data-transformation specifications, showing the control word and the specifications text delimited by apostrophies, would be

TRANSFØRMATIONS ' X(2) + X(3) = X(51) SQRTF X(2) = X(2) '.

In most SPAN modules, data transformations are declared by the control word TRANSForm. In the STARS Files Tabulator and STARS File Transformation modules, where transformations can be expressed on results of certain subsequent processes in addition to those expressed on the input data, control word TRANsform declares input data transformations and the word F-TRANsform is reserved for transformations on data resulting from subsequent processes. In the STARS Files Collator module, where separate transformation specifications

can be declared on data from any one of three possible source files, control words 1)TRANsform, 2)TRANsform, 3)TRANSform introduce the first, second, and third file-data transformations respectively. Definitive information on control words usage in a particular program is given in the SPAN Control Words Glossary, TM-1563/013/xx.

Within the actual specifications text, blanks and commas are ignored. Terms and statements need not be separated in any special way; or, if desired, statements may be separated by commas. Different transformations may appear on the same card, or on separate cards indented for readability. For example, the following two sets of specifications are equivalent:

```
TRANSFØRM 'X(5)+(3.)EQX(29)GØTØ$1$LØGX(13)=X(13)GØTØ
$2$$1$LØGX(14)=X(13)$2$X(1)-X(2)=X(1004)'.

TRANSFORM  ' X(5) + (3.) EQ X(29) GØ TØ $1$
             LØG X(13) = X(13), GØ TØ $2$
        $1$  LØG X(14) = X(13),
        $2$  X(1) - X(2) = X(1004)      '.
```

According to the rules for constructing control specifications in SPAN (see TM-1563/012/xx), it is possible to intersperse comments and other optional text among the elements of a control sentence. It may be also desirable to associate explanatory notes with individual statements in the data-transformation text. The following example illustrates this flexibility:

```
CALCULATE THE FØLLØWING *TRANSFØRMS OF INPUT DATA ---

'X(3) / X(25) = X(1127)'          GRØSS PØPULATIØN DENSITY
'X(20)-X(25)=X(600), X(3)/X(600)= X(1128)' NET RES.DENSITY.
```

If one prefers, for clarity, to write transformation statements on separate cards, enclosing the statements by apostrophies may serve yet another purpose. The text of data-transformation specifications input to a single SPAN job may not exceed 6000 characters, including blanks and commas. By enclosing individual statements in apostrophies and, thus, purging the specifications text of unnecessary characters, a maximum of meaningful text can be accommodated in a control sentence without sacrificing readability. Care must be taken, however, that the control sentence contain an <u>even</u> number of the delimiting apostrophies.

DATA TRANSFORMATIONS: LIMITATIONS

Data-transformation specifications input to the STARS Files Collator module may not contain any arithmetic terms; that is, transformations may be performed on I-region data only. No similar restriction applies to specifications input to other SPAN modules.

The size of internal tables used in the translation of the data-transformation specifications sets limits, in a particular SPAN job, on (1) the length of the specifications text, (2) the number of transformation terms permitted, and (3) the number of constants permitted. These limits are shown in Figure II-8. Appropriate diagnostics are printed if these limits are exceeded. In addition, space allocated to receive the source data and data transforms, the I-region and X-region, is of defined size. References beyond the limits of the I-region or X-region as defined in Figure II-8, i.e., references such as X(6073) or I(375), are not detected.

In all SPAN modules, the data-transformation text associated with a control word may not exceed 6000 characters in length. (In the STARS Files Collator, where an independent set of data transformations can be specified on each of three possible source files, this limit applies to each set of specifications separately.)

DATA TRANSFORMATIONS: LIMITATIONS

Module	I-Region Size (no. of elements)	X-Region Size (no. of elements)	Maximum Transformation Terms (no. of terms)	Maximum Constants (no. of constants)
Factor Analysis	100	400	500	100
Regression Analysis	100	400	500	100
Rectangular Product Moments	100	1400	1000	200
File Transformation	100	1600	1000*	200*
STARS Files Abstractor	100	1400	1000	200
STARS Files Collator	100**	1000**	450***	90***
STARS Files Tabulator	100	1400	1000*	200*
STARS Files Summary-Sort	100	1400	1000	200

*Combined total for input and subsequent process data transformations.
**For each of three possible source files.
***Combined total for all source files.

Figure II-8

The maximum of transformation terms permitted refers to simple terms. For the purpose of calculating this maximum, the SUM and MED complex terms are equivalent to two and three simple terms respectively. The GO TO term, whether appearing in a condition statement or an unconditional transfer statement, counts as a simple term. The comparison operator combined with the first term of the right-hand expression in a condition statement counts as a simple term.

The maximum of constants permitted refers to each actual appearance of a constant in an expression. Statement labels do not enter into the count of terms or constants. There is no practical limit on the number of labels.

TRANSFORMATIONS TEXT ERROR DETECTION AND DIAGNOSTICS

Before a file is actually processed, the data-transformation specifications text is translated into an internal representation designed for more efficient execution. The translator detects certain errors in the specifications text and prints appropriate diagnostic messages. Errors detected are those relating to (1) excessive size of the transformations text, (2) use of illegal operators and operands, (3) violation of restrictions on the number of elements of a specifications text, and (4) incorrect labeling of statements. Other violations of data-transformation specification rules, such as use of mixed expressions, are not detected. A detected error in data-transformation specifications renders a SPAN job non-executable. Control, in that case, passes to the next job in line.

Diagnostic messages which may result during processing of data-transformation specifications are listed below, with comments and examples where appropriate.

Excessive Size of Transformations Text

*** DATA-TRANSFORMATION SPECIFICATIONS TEXT IS TOO LONG (OVER 6000 CHARACTERS).

This diagnostic message results from violation of the requirement that the transformations text associated with a control word not exceed the maximum length of 6000 characters. In calculating the length of the transformations text, note that each string-terminating apostrophe used in the control sentence may cause 1-5 blank characters to be appended to the string to maintain the internal string length as an integral multiple of six characters. Blanks so generated would contribute to the character count of the string.

Illegal Use of Operators and Operands

OPERATOR IS NOT IN OPERATIONS TABLE.

An illegal operator appears in the specifications text, as shown in the following example:

*** TRANSFORMATIONS PROGRAM DIAGNOSTIC
OPERATOR IS NOT IN OPERATIONS TABLE.

ERROR SENSED WHILE SCANNING THE UNDERLINED WORD IN THE FOLLOWING PORTION OF THE SOURCE TEXT ---

XI(1) * XI(15) TØ XI(24) +

[TØX] is not a legal integer term operator (see Figure II-6).

FIRST CHARACTER OF A CONSTANT IS ILLEGAL.

The first character of a constant is a non-numeric character other than H, Ø, or - (minus) sign; for example,

(GABC), ((HHAPPY).

ILLEGAL CHARACTER IN AN OCTAL CONSTANT.

Octal constant contains a character other than digits 0 through 7; for example,

(Ø 380).

ILLEGAL CHARACTER IN A DECIMAL CONSTANT.

Decimal number or integer constant contains a character other than the - (minus) sign or digits 0 through 9; for example,

(136A).

Violation of Certain Restrictions

TOO MANY OPERATIONS SPECIFIED IN SOURCE TEXT.

The maximum number of terms permitted in the specifications text has been exceeded. See Figure II-8 concerning limits in various modules.

TOO MANY CONSTANTS SPECIFIED IN SOURCE TEXT.

The maximum number of constants permitted in the specifications text has been exceeded. See Figure II-8 concerning limits in various modules.

Labeling Errors

DUPLICATE TRANSFORMATIONS LABELS.

More than one statement has been assigned the same label.

MISSING TRANSFORMATIONS LABELS AS FOLLOWS-- [statement label$_1$]
[statement label$_2$]
:::::::::
[statement label$_i$]

EXECUTION ERROR DETECTION AND DIAGNOSTICS

Certain error conditions may arise during execution of the data-transformation specifications. Where, in an expression, the function of a variable would not yield a real number of allowable magnitude, or where there is division by zero, the value of the entire expression is automatically set to -0. These conditions do not result in diagnostic comment. Certain floating-point operations (in arithmetic terms) when operating on data that are not floating-point type may result in internal register underflow or overflow. A diagnostic message is printed if such a condition should occur. The results in that case are suspect and the source file should be examined for bad data.

EXAMPLES OF DATA TRANSFORMATIONS

In the following, three examples of data transformations are given. The first example represents a complete transformation program. The second example shows how to specify the computation of a median. The third is an example of the use of the weighting function.

Example I

```
PERFØRM THE FØLLØWING *TRANSFØRMATIONS,

          'X(1) + X(2) / X(11) =X(1001)
           X(1) * X(5) = X(1002)
           X(4) + X(7)  LQ  (5.) GØ TØ $1$
           X(4) + X(7) LQ  (20.) GØ TØ $2$
           (0.) = X(1003)
           GØ TØ $STØP$
$1$ X(38)/ X(39) + X(4) = (1003)
           GØ TØ $STØP$
$2$ X(40)/ X(39) + X(4) = X(1003)
$STØP$
```

Note that the last labeled statement may be "empty" (label $STØP$), permitting a transfer to the end of the transformations specifications.

(Page 24 Blank)

Example II

Assume that the "median of school years completed" is to be calculated for a population distribution by number of school years completed. The following is the layout of information in the X-region:

variable	location
no. of persons, 0-6 school yrs. completed	X(10)
no. of persons, 6-9 school yrs. completed	X(11)
no. of persons, 9-12 school yrs. completed	X(12)
no. of persons, 12-14 school yrs. completed	X(13)
no. of persons, 14-16 school yrs. completed	X(14)

To compute the median of the values in X(10) through X(14) it is necessary to have available in the X-region, in consecutive locations, each of the lower limits of the classification intervals, e.g., 0, 6, 9, 12 and 14 and also the upper limit of the last interval, e.g., 16, which could be a "dummy" value. Notice that in the transformation statements below the assignment of these numbers to specific, although arbitrary, X-region locations is shown.

```
TRANSFØRM  'X(251) EQ (6.) GØ TØ $MED$
           (0.) = X(250) (6.0) = X(251) (9.) =X(252)
           (12.) =X(253) (14.) =X(254) (16.) =X(255)
     $MED$ MED X(10) THRU X(14) INT X(250) =X(1001)'.
```

Note that MED is an operator, but MED is a label. As the first entity is processed, X-region locations X(250) to X(255) contain zeros. The statements above set the locations to the desired quantities. After the first entity there is a 6 in X(251) so that, as a result of the first statement, the statements setting up the interval locations need not again be executed; instead, the program immediately proceeds to MED.

Example III

The weighting function has been previously defined, and has several uses. One possible use is illustrated below, in which, to save writing transformation specifications, instead of:

```
X(69) * X(208) = X(1001)
X(70) * X(208) = X(1002)        etc.
```

one could write:

```
X(69) = X(1001)
X(70) = X(1002)                 etc.
WGHT X (1001) THRU X(1---) BY X(208)
```

If only one variable is to be weighted and label is to be preserved, then one might use WGHT X(n) THRU X(n) BY X(m) where n and m are variable indexes.

III. ENTITY STRATIFICATION

The stratification facility in SPAN programs allows the user to select or stratify records from an input file by stating explicit conditions on the properties associated with an entity. These condition statements are evaluated during the input processing of each entity record (see Figure I-1). If a stated condition is true, the entity is assigned the specified stratum number. If the stated condition is false, the next stated condition is evaluated. If none of the stated conditions are true, the stratum number is automatically set at zero; the entity record is thus rejected from further processing, and the next entity record is obtained, as shown in Figure I-1.

Stratification specifications are executed after transformations have been performed on the input data. Thus conditions may be stated on both input variables and their transforms. The stratum number assigned to the entity is placed into I(99) and is therefore available to subsequent processes.

Entity stratification specifications could be expressed as a series of condition and transformation statements. The following transformation specifications, for example, will assign the stratum number 8 to entities that contain a 3, or a 5 through 10, in I(3):

```
          I(3) EQ (3) GØ TØ $CLASSF$
          I(3) GQ (5) GØ TØ $NEXT$
          GØ TØ $DELETE$
$NEXT$    I(3) LQ (10) GØ TØ $CLASSF$
$DELETE$  (0) = I(99) GØ TØ $END$
$CLASSF$  (8) = I(99)
                $END$
```

The same stratification can be accomplished with the following stratification specifications:

```
ASSIGN 8 IF I(3) = 3, 5-10.
```

Thus, one can express more concisely a complex set of conditions with respect to sets of explicit values. However, transformation specifications permit the use of variables and expressions as condition criteria.

STRATIFICATION STATEMENTS

Entity stratification specifications are composed of an arbitrary collection of statements. These statements, called stratification statements, are made up of condition clauses, and these, in turn, of criterion values. These and other elements of the entity stratification specifications are discussed below.

General Form
ASSIGN n IF c_1 ,AND c_2 ... ,AND c_i. where n is a stratum number defined below, and $c_1, c_2, \ldots, c_i$ are condition clauses defined below.

A stratification statement assigns a stratum number to an entity if the condition expressed by the statement is true. The statement may contain more than one condition clause, in which case the clauses are connected in logical-and fashion by the [,AND] operator. A stratification statement always terminates on a period [.] .

The stratum number must be an unsigned integer not exceeding, in most SPAN programs, 2^{15} in magnitude. In the STARS Files Tabulator, however, the maximum stratum number is 80.

A condition clause asserts the equality of the value of a variable or code to at least one of a set of criterion values.

General Form
p = v where p is a code or variable, and v is a set of criterion values of a form described below.

For example,

I(3) = 3, 5-10, 17, 28-30

asserts that I(3) equals 3, 5 through 10, 17, or 28 through 30.

Elements of the condition clause are criterion values or ranges of criterion values, as shown. The elements are connected by commas [,] in logical-or fashion. The limits of the range are connected by the [-] symbol and may be stated in any order, e.g., 28-30 could be expressed as 30-28.

The criterion values may be decimal constants (D), decimal integer constants (K1), octal integer constants (K2), or alphameric constants (K3), where the symbols in parentheses are those used in Figure II-1, Chapter II, (and defined in that same chapter). Note that the following element is possible: -.1--.3, for -.1 through -.3. The statement of a range between two criterion values is interpreted as the range between their binary code values. In particular, HABC-HABD is equivalent to ∅212223606060-∅212224606060. In stratification text, constants are not enclosed in parentheses.

The criterion values for a code, e.g., I(72), must be of the type K1, K2, or K3; the criterion values for a variable, e.g., X(51), must be of the type D.

The following are examples of stratification statements:

ASSIGN 2 IF I(75) = 3, 5, 8-10,HJUNK, AND X(10) = -.2-55.3.

ASSIGN 185 IF X(10) = -35.--36., 39..

In the second example, all periods but the last are interpreted as decimal points. The final period terminates the statement. Minus [-] symbols play a similar dual role: the first and third [-] signs denote negative numbers, while the second [-] sign defines the range between -35. and -36..

HOW TO SPECIFY ENTITY STRATIFICATION

Entity stratification specifications that are to operate on the data input (including data transformations, if any) to a particular SPAN job must be included as a separate sentence among the control specifications for that job. In a control sentence, the specifications text takes the form of a string predicate of an appropriate control word; that is, the text is delimited by apostrophies and preceded by a stratification- or selection-declaring control word.

General Form
SELECT STRATIfy ⊕)SELEct } s where s is the entity-stratification specifications text (S6000), and ⊕ stands for source file number in the case of multifile input (⊕ = 1,2,3 in the STARS Files Collator).

A simple example of entity-stratification specifications, showing the control word and the specifications text delimited by apostrophies, would be

SELECT THE FØLLØWING ØBSERVATIØNS ' ASSIGN 1 IF X(20)= .0 . '.

In most SPAN modules stratification is used to select records from the source file, and for these modules the control word is SELECT. In the STARS File Tabulator, the purpose is stratification, and the control word is STRATIFY. In the STARS File Collator where the selection can be specified on data from any one of three possible source files, control words 1)SELEct, 2)SELEct, and 3)SELEct introduce the first, second, and third file entity-stratification text respectively. Information on control words usage in a particular module is given in the SPAN Control Words Glossary, TM-1563/013/xx.

ENTITY STRATIFICATION: LIMITATIONS

The size of internal tables used in the translation of the entity-stratification specifications sets limits, in a particular SPAN job, on (1) the length of the specifications text, (2) the maximum size of the stratum number, (3) the number of condition clauses permitted, and (4) the number of criterion values permitted. Appropriate diagnostics are printed if these limits are exceeded.

In all SPAN modules, the entity-stratification specifications text associated with a control word may not exceed 6000 characters in length. (In the STARS Files Collator, where entity stratification may be specified independently on each of the three possible source files, this limit applies to each set of specifications separately.)

Other limits vary by module as shown in Figure II-9.

ENTITY STRATIFICATION: LIMITATIONS			
Module	Maximum Stratum Number	Maximum Condition Clauses	Maximum Criterion Values
Factor Analysis	2^{15}	20	100
Regression Analysis	2^{15}	20	100
Latent Class Analysis	2^{15}	20	100
Rectangular Product Moments	2^{15}	20	100
File Transformation	2^{15}	20	100
STARS Files Abstractor	2^{15}	20	100
STARS Files Collator	2^{15}	45*	450*
STARS Files Tabulator	80	80	400
STARS Files Summary-Sort	2^{15}	80	400
*Combined total for all source files.			

Figure II-9

In determining the number of criterion values used in a specifications text note that each expressed range of values counts as a single value.

STRATIFICATION TEXT ERROR DETECTION AND DIAGNOSTICS

Before a file is processed, the entity-stratification specifications text is translated into an internal representation designed for more efficient execution. The translator detects certain errors in the specifications text and prints appropriate diagnostic messages. Errors detected are those relating to (1) excessive size of the stratification text, (2) illegal statement syntax, and (3) violation of restrictions on the number of elements of a specification text. An error detected in entity-stratification specifications renders a SPAN job non-executable. Control in that case passes to the next job in line.

Diagnostic messages that may result during translation of entity-stratification specifications are listed below, with appropriate comments and examples.

Excessive Size of Stratification Text

*** ENTITY SELECTION SPECIFICATIONS TEXT IS TOO LONG (OVER 6000 CHARACTERS).

*** ENTITY STRATIFICATION SPECIFICATIONS TEXT IS TOO LONG (OVER 6000 CHARACTERS).

These messages result from violation of the requirement that the stratification text associated with a control word not exceed the maximum length of 6000 characters. In calculating the length of the stratification text, note that each string-terminating apostrophe used in the control sentence may cause 1-5 blank characters to be appended to the string to maintain the internal string length at an integral multiple of six characters. Blanks so generated would contribute to the character count of the string.

Illegal Statement Syntax

WHILE EXPECTING AN -IF- OR -AND- WORD, PROGRAM HAS ENCOUNTERED A WORD IT CANNOT RECOGNIZE.

A condition clause is preceded by an illegal operator, as shown in the following example:

*** STRATIFICATION PROGRAM DIAGNOSTIC

WHILE EXPECTING AN -IF- OR -AND- WORD, PROGRAM HAS ENCOUNTERED A WORD IT CANNOT RECOGNIZE.

ERROR SENSED WHILE SCANNING THE UNDERLINED WORD IN THE FOLLOWING PORTION OF THE SOURCE TEXT ---

3,78-92,AND IF X(42)=

[,AND IF] is not a legal clause connector. Similar error message would result if a stratification statement began with [ASSIGN 56 X(22)=] .

ILLEGAL CHARACTER IN AN OCTAL CONSTANT.

Octal constant contains a character other than digits 0 through 7; for example, ø 380.

ILLEGAL CHARACTER IN A DECIMAL CONSTANT.

Decimal number or integer constant contains a character other than the minus [-] sign, (e.g., 3AB, 57+2) or digits 0 through 9.

DECIMAL INTEGER IN A DECIMAL NUMBER CRITERION SET.

A decimal number criterion set contains a constant without a decimal point [.] , as, for example, in the following text:

3.5,6.,.5-1.,1, AND X(7) =

Violation of Certain Restrictions

TOO MANY CONDITION CLAUSES IN SOURCE TEXT.

The maximum number of condition clauses permitted in the specifications text has been exceeded. See Figure II-9 concerning limits in various modules.

TOO MANY CRITERION VALUE SETS IN SOURCE TEXT.

The maximum number of criterion values permitted in the specifications text has been exceeded. See Figure II-9 concerning limits in various modules.

Certain other violations of specification rules for entity stratification are not detected directly. For instance, a decimal point [.] appearing in a decimal integer constant would be interpreted as a period [.] terminating the statement. If the last statement is not properly terminated by a period, that statement will be translated incorrectly, and the stratification specifications will not execute properly. A criterion value beginning with a non-numeric character other than ∅ will be automatically interpreted as an alphameric constant, unless, of course, it is contained in a decimal number criterion set.

EXAMPLES OF ENTITY STRATIFICATION

Three examples of entity stratification are given below. The first example shows the straightforward use of stratification specifications for selecting entity records for the source file. The second and third examples demonstrate the joint use of data transformations and entity stratification.

Example I

Assume a source file contains 16 records. Code I(2) is the record sequence number. Values of variable X(210) range from -50.7 to +200.5.

Records 4, 6 through 8 and 12 in the source file are to be assigned stratum number 11. Records sequence-numbered 1, 5, 13, and 14 and containing values of X(210) between 0. and -50.7 are to be assigned stratum number 12. Records sequence-numbered 1, 5, 13, and 14 and containing values of X(210) between 0. and 200.5 are to be assigned stratum number 13.

The following specifications will perform the desired stratification:

```
SELECT   'ASSIGN 11   IF   I(2)    = 4,6-8,12.
          ASSIGN 12   IF   I(2)    = 1,5,13,14, AND
                           X(210)  = 0. -- 50.7.
          ASSIGN 13   IF   I(2)    = 1,5,13,14, AND
                           X(210)  = 0. - 200.5. '.
```

As a result, records sequence-numbered 2, 3, 9, 10, 11, 15, and 16 will not satisfy any of the stated conditions and will be excluded from further processing. The stratum number values are placed in I(99) and may be used in further processing, e.g., as a key for sorting or summarizing the derivative file.

Example II

This example shows the joint use of data transformations and entity stratification. Records are to be selected on a source file code, I(2) and if a computed percentage exceeds 50%.

```
TRANSFØRMATIØNS   ' SUM X(3) THRU X(16) = X(1001)
                    X(13) / X(1001) * (100.) = X(1002) '.

SELECT   'ASSIGN 1 IF I(2)=1-5, AND X (1002) = 50.-100. .'.
```

Example III

The following example again illustrates the use of data transformations in conjunction with entity stratification. The purpose here is to delete from the derivative file all records possessing duplicate values of a key, I(2).

```
TRANSFORM   ' I(2) EQ I(60) GØTØ $1$
              (0) = I(61) GØTØ $2$
        $1$   (1) = I(61)
        $2$   I(2) = I(60)  '.

SELECT   'ASSIGN 1 IF I(61) = 0.  '.
```

APPENDIX A

7090 REPRESENTATION OF ALPHAMERIC CHARACTERS

Character	Card	BCD Tape	Storage	Character	Card	BCD Tape	Storage	Character	Card	BCD Tape	Storage	Character	Card	BCD Tape	Storage
1	1	01	01	A	12 1	61	21	J	11 1	41	41	/	0 1	21	61
2	2	02	02	B	12 2	62	22	K	11 2	42	42	S	0 2	22	62
3	3	03	03	C	12 3	63	23	L	11 3	43	43	T	0 3	23	63
4	4	04	04	D	12 4	64	24	M	11 4	44	44	U	0 4	24	64
5	5	05	05	E	12 5	65	25	N	11 5	45	45	V	0 5	25	65
6	6	06	06	F	12 6	66	26	O	11 6	46	46	W	0 6	26	66
7	7	07	07	G	12 7	67	27	P	11 7	47	47	X	0 7	27	67
8	8	10	10	H	12 8	70	30	Q	11 8	50	50	Y	0 8	30	70
9	9	11	11	I	12 9	71	31	R	11 9	51	51	Z	0 9	31	71
blank	blank	20	60	+	12	60	20	-	11	40	40	0	0	12	00
=	8-3	13	13	.	12 8-3	73	33	$	11 8-3	53	53	,	0 8-3	33	73
'	8-4	14	14	)	12 8-4	74	34	*	11 8-4	54	54	(	0 8-4	34	74

This table has been reprinted by permission from IBM 7090/7094 Programming Systems FORTRAN II Programming, Form No. C28-6054-5.

MODIFICATION TO:

TM-1563/014/01, "SPAN REFERENCE MANUAL: SPAN Data-Transformations and Stratification Capability," 4 March 1965.

APPROVED

V. V. Almendinger

Vladimir V. Almendinger

System Development Corporation / 2500 Colorado Ave. / Santa Monica, California

CURRENT MODIFICATION*

Modified Pages	Notes and Filing Instructions
7-8	Remove page 7-8 dated 4 March 1965 and replace with page 7-8 dated 24 September 1965.
9-10	Remove page 9-10 dated 4 March 1965 and replace with page 9-10 dated 24 September 1965.
11	Remove page 11-12 dated 4 March 1965 and replace with page 11-12 dated 24 September 1965.

*The modified portions of the text are indicated by a double verticle bar opposite the text.

FUNCTIONS OF VARIABLES	
Square root	SQRT X(90)
Logarithm (natural)	LØG X(90)
Exponential	EXP X(90)
Sine	SIN X(90)
Cosine	CØS X(90)
Arctangent	ATAN X(90)

Figure II-2

The function of a variable is a single operand. Where the function of a variable would not yield a real number of allowable magnitude, as in $(-4)^{\frac{1}{2}}$, log 0, or e^{90}, the value of the function is set to -0.

DECIMAL NUMBER CONSTANTS (D)

General Form
A decimal number constant consists of a string of decimal digits with a decimal point at the beginning, at the end, or between two digits.

The decimal number constant may be signed or unsigned. A decimal number is represented internally as a floating-point number. The magnitude of a decimal number constant must lie between 10^{-14} and 10^{14}, or be zero. Examples: (-.15), (3.1416), (12.) or (0.).

DECIMAL INTEGERS (C1 or K1)

General Form
A decimal integer constant consists of 1-5 decimal digits written without a decimal point. A decimal integer code is represented by I(n), where n is the index of its location in the I-region.

A decimal integer may be signed or unsigned. Its magnitude must be less than 2^{15}. A decimal integer constant would be expressed as (1492), (-2), or (0). A decimal integer code is represented by its address, for example, I(92). Decimal integers are stored internally as pure binary numbers, occupying the left-half word, with the right 18 binary bits being all zeros. If a decimal integer is negative, the minus sign occupies the high-order position of the binary word; this must be remembered when using negative integers in Boolean operations.

OCTAL INTEGERS (C2 or K2)

General Form
An octal integer constant consists of 1-12 octal digits preceded by the letter O. An octal integer code is represented by I(n), where n is the index of its location in the I-region.

The maximum value of an octal integer is 2^{36}. Leading zeros need not be indicated. Examples: (Ø356), (Ø777700007777), (Ø3). An octal integer code is represented by its address, for example, I(92). Octal integers are stored internally as pure binary numbers, each octal digit comprising three binary bits.

ALPHAMERIC CONSTANTS AND CODES (C3 or K3)

General Form
An alphameric constant consists of 1-6 alphabetic and/or numeric characters preceded by the letter H. An alphameric code is represented by I(n), where n is the index of its location in the I-region.

An alphameric constant may not include special characters, but may include blank spaces to be treated as characters. The H to indicate mode must be used even if word begins with an H. Examples: (HABC), (H2376), (H23KV6), (HHAPPY). If the string exceeds six characters, only the six left-most characters are used. For example, (HALPHAMERIC) is equivalent to (HALPHAM), i.e., the first six characters. Blank spaces occurring within the six character positions following an H are treated (and counted) as characters.

An alphameric code is represented by its address, for example: I(92). Alphameric words are stored internally as binary numbers, with two octal digits, and therefore six binary bits, for each character. They are left-justified, so that the right characters consist of blanks. The IBM 7090 octal (storage) representation of alphameric characters is given in Appendix A , "7090 Representation of Alphameric Characters."

COMPLEMENT OF OCTAL INTEGER CODE (C4)

The 1's complement of an octal integer code is indicated by placing a [-] symbol ahead of the address of a code, for example, -I(15). The combination represents a single operand.

ARITHMETIC TERMS

Arithmetic terms may be simple or complex. A simple arithmetic term consists of one operator and one operand. A complex arithmetic term represents a single operation performed on more than one operand. Figure II-3 lists the operators that may be used in simple arithmetic terms; Figure II-4 lists the two types of complex terms. Only certain operands may be used with a given operator; these are indicated in Figures II-3 and II-4 by means of the operand symbols defined in Figure II-1.

SIMPLE ARITHMETIC TERM OPERATORS		
Operator	Operation	Applicable to Operand Types
+	Add	V, F, D
-	Subtract	V, F, D
*	Multiply	V, F, D
/	Divide	V, F, D
TØ	Store and proceed	V
=	Store and clear	V

Figure II-3

Division by zero will set the quotient to -0. The difference between the two "store" operators will be discussed in the section on expressions.

COMPLEX ARITHMETIC TERMS		
Term	Operation	Admissible Operand Types
SUM x_1 THRU x_2	Sum operands x_1 through x_2	V
MED x_1 THRU x_2 INT x_3	Calculate median over variables x_1 through x_2, with x_3 being the first element of a set of the lower limits of corresponding category intervals; if n is the number of categories, the (n+1)th element contains the upper limit of the nth interval.	V

Figure II-4

A detailed example of a median computation will be given later.

BOOLEAN TERMS

Boolean terms consist of one operator and one operand. They are listed in Figure II-5. Only certain operands may be used with a given operator; these are indicated in Figure II-5 with the operand symbols defined in Figure II-1.

BOOLEAN TERM OPERATORS		
Symbol	Operation	Applicable to Operand Types
+	logical-or	All C and K2, K3
*	logical-and	All C and K2, K3
/	exclusive-or	All C and K2, K3
LEFT	shift left n bits	K1
RIGHT	shift right n bits	K1
TØ	store and proceed	All C
=	store and clear	All C

Figure II-5

For Boolean operations, including shifts, recall that all operands are internally represented as pure binary quantities of 36 binary bits. Shifts are, in effect, accumulator shifts, not "long" or "logical" shifts; therefore bits are lost in this process. On a left shift of n bits, (n-1) bits are lost, while n bits are lost on a right shift of n bits. Vacated positions are filled with zeros.

INTEGER TERMS

Integer terms consist of an operator and an operand. The operators, and applicable operands, are given in Figure II-6.

INTEGER TERM OPERATORS			
Applicable to Operand Type C1 (Decimal Integer Code)		Applicable to Operand Type K1 (Decimal Integer Constant)	
Symbol	Operation	Symbol	Operation
+X	Integer Code Add	+	Integer Constant Add
-X	Integer Code Subtract	-	Integer Constant Subtract
*X	Integer Code Multiply	*	Integer Constant Multiply
/X	Integer Code Divide	/	Integer Constant Divide
=X	Store and Clear		

Figure II-6

EXPRESSIONS

An expression is a string of terms, none of which contains the operator [=] or [=X]. Each line below is an example of one expression:

Expression	Type
X(52) - X(43) / (2.84)	Arithmetic
I(2) * (Ø770000) LFT (18) + (HØUKE)	Boolean
(.55) - SQRTX(55) - X(12)	Arithmetic
(Ø707) / I(87) + -I(3)	Boolean
XI(52) /X I(33) *X I(1)	Integer
MED X(14) THRU X(21) INT X(38) TØ X(80) * (2.)	Arithmetic

Blanks are ignored in expressions, so that expressions equivalent to the above can be written in many ways. In the above examples, each of the following represents a single term: -X(43); LEFT (18); - SQRTX(55); - -I(3); /X I(33); TØ X(80); MED X(14) THRU X(21) INT X(38).

RULES FOR CONSTRUCTING EXPRESSIONS

1. All terms in an expression must be of the same type; e.g., all arithmetic terms, Boolean terms, or integer terms. Mixed expressions are not detected by the translator and will yield incorrect results.

2. A complex arithmetic term may appear only as the leading term. Thus the following is a correctly formed expression,

   SUMX(35) THRU X(40) * X(2)/(100.)

   while the following expression is in error,

   X(2) * SUMX(35) THRU X(40)/(100.)

3. For expressions made up of either simple arithmetic or Boolean terms the operator of the leading term is always an implicit [+] . That is, no operator except the [+] may be used for the leading term, and the [+] is never written. (See the previously given examples.)

4. Expressions are evaluated from left to right; the operator symbol in each term defines the relation between the result of preceding terms and the value referred to by the operand. In the expression X(52) - X(43) / (2.84) the quantity X(43) is subtracted from X(52) and the result is then divided by (2.84). If one wanted to compute X(43) divided by (2.84) and the quotient to be then subtracted from X(52) one would need to first perform the division and then store the quotient in a temporary location, as will be discussed in the section on Statements. In some cases, other methods can be used; thus, if the operator ahead of X(43) were a + instead of a -, one could write X(43) / (2.84) + X(52). In general, if $\oplus$ is a typical operator and x is a typical operand, then the expression

   $$x_1 \oplus_2 x_2 \oplus_3 x_3 \oplus_4 x_4$$

   is evaluated as if it read

   $$(((x_1 \oplus_2 x_2) \oplus_3 x_3) \oplus_4 x_4)$$

   However, parentheses may not be used for this purpose in an expression.

DOCUMENT CONTROL DATA - R&D

1. ORIGINATING ACTIVITY (Corporate author) System Development Corporation, Santa Monica, California	2a. Unclassified 2b

3. REPORT TITLE

SPAN REFERENCE MANUAL SPAN DATA-TRANSFORMATIONS AND STRATIFICATIONS CAPABILITY.

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (Last name, first name, initial)

Almendinger, V. V.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
24 September 1965	7	

8a. CONTRACT OR GRANT NO. CPR-11-1543, for the Bureau of Public Roads, U. S. b. PROJECT NO. Department of Commerce c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) TM-1563/014/01A 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. AVAILABILITY/LIMITATION NOTICES

This document has been cleared for open publication and may be disseminated by the Clearing House for Federal Scientific & Technical Information.

11. SUPPLEMENTARY NOTES Modifies TM-1563/014/01, by V. V. Almendinger, dated 4 March 1965, DDC number: AD-613 285	12. SPONSOR

13. ABSTRACT

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
SPAN System Data Management Statistical Analysis IBM 7090 Computer IBM 7094 Computer Social Science Urban Data Analysis						